UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

David Mohorčič

Analiza platform v oblaku za potrebe integracije z mobilnimi aplikacijami

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2015

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

David Mohorčič

Analiza platform v oblaku za potrebe integracije z mobilnimi aplikacijami

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Damjan Vavpotič

Ljubljana, 2015

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Analiza platform v oblaku za potrebe integracije z mobilnimi aplikacijami

Tematika naloge:

V okviru diplomske naloge se osredotočite na vidik vzpostavitve povezave med platformami v oblaku in mobilnimi napravami z operacijskim sistemom Android. Na začetku kratko predstavite ključne tehnologije, nato pa analizirajte in predstavite nekaj konkretnih primerov platform v oblaku. Za dve izmed analiziranih oblačnih platform izdelajte delujoča prototipa zalednega sistema, ki se bosta povezala s prav tako delujočim prototipom mobilne aplikacije. Kritično ovrednotite in primerjajte tako nastala prototipa sistema kot tudi sam potek razvoja na obeh izbranih oblačnih platformah. Predstavite prednosti in slabosti dela na vsaki od uporabljenih oblačnih platform.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani David Mohorčič, z vpisno številko **63100116**, sem avtor diplomskega dela z naslovom:

Analiza platform v oblaku za potrebe integracije z mobilnimi aplikacijami

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Damjana Vavpotiča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) enaki tiskani obliki diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 5. marec 2015 Podpis avtorja:

Zahvalil bi se družini za vso pomoč in podporo v času mojega šolanja. Zahvaljujem se mentorju dr. Damjanu Vavpotiču za nasvete in usmerjanje pri izdelavi naloge, ter za potrpežljivost ob mojem delu. Zahvalil bi se tudi punci Kristini, ki me je spodbujala in podpirala v študijskem obdobju.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	Application programming interface	Vmesnik uporabniškega program
GB	Gigabyte	Gigabajt
GPS	Global positioning system	Sistem globalnega pozicioniranja
HTTP	Hyper text transfer protocol	Hipertekstovni prenosni protokol
IAAS	Infrastructure as a service	Infrastruktura kot storitev
JSON	JavaScript object notation	oblika za izmenjavo podatkov
MHz	Megahertz	Megaherc
OS	Operating system	Operacijski sistem
PAAS	Platform as a service	Platforma kot storitev
REST	Representational state transfer	Arhitektura za izmenjavo podatkov med spletnimi storitvami
SAAS	Software as a service	Aplikacije kot storitev
SDK	Software development kit	Komplet programskih orodij za razvijanje programske opreme
SMS	Short message system	Sistem kratkih sporočil
SQL	Structured query language	Strukturirani poizvedovalni jezik za delo s podatkovnimi bazami
XML	Extensible markup language	Razširljiv označevalni jezik

Povzetek

Diplomska naloga je namenjena primerjavi povezav mobilne aplikacije z različnimi ponudniki storitev v oblaku. Najprej je predstavljenih šest ponudnikov platforme v oblaku. Za vsakega posebej lahko preberemo, kaj ponujajo v brezplačni ponudbi in katere vire lahko koristimo pri plačljivi licenci. Za lažjo primerjavo sta priloženi tudi tabeli. Skupno vsem ponudnikom je, da podpirajo povezavo z aplikacijo, ki je razvita za operacijski sistem Android. Naša aplikacija beleži in prikazuje podatke o raznih dogodkih, ki jih prejemamo prek SMS sporočil. Zaradi velike količine podatkov se le-ti arhivirajo v podatkovne strukture oblaka. Aplikacija omogoča arhiviranje v Googlove ali Microsoftove podatkovne baze, za katere je prikazana tudi praktična uporaba storitev v oblaku.

S primerjavo ponudnikov smo pridobili informacije, kaj lahko pričakujemo, ko se odločimo za uporabo oblaka v naših programskih rešitvah. Izkazalo pa se je tudi, da lahko Android aplikacija dodobra izkoristi vire v oblaku.

Ključne besede: platforma v oblaku, Android povezan z oblakom, Google app engine, Microsoft Azure

Abstract

The main purpose of this diploma dissertation is to compare connections of mobile application with different cloud service providers. At the beginning there are represented six providers of cloud platforms. For each of them we could read what is included within free licence and which sources could be used in payable licence. To make a comparison easier there are also two tables. What is common to all providers is that their service could be connected to the application that is made for Android operating system. Our mobile application is making notes and showing different events that come through text messages. Because of the great amount of data, all data is archived in cloud's data structures. Application enables archiving in Google's and Microsoft's databases of which also practical usage is shown.

With this comparison we have got information of what we can expect when we decide to use cloud service for our software solutions. We found that Android application could utterly use cloud's sources.

Keywords: platform as a service, Android connect with cloud, Google app engine, Microsoft Azure

Poglavje 1 Uvod

Razvoj tehnologije na vseh področjih hitro napreduje in prinaša nove rešitve. Tako je tudi v računalništvu. Na začetku so bili podatki dostopni v lokalnih omrežjih in uporabniki so bili tako omejeni z dostopom do njih. Vedno večja je bila težnja po dostopanju od koder koli in kadar koli. Tako se je v svetu uveljavil pojem računalništva v oblaku (ang. cloud). Razvijati so se začele nove metode, ki uporabljajo vire v oblaku. Ta se v splošnem ne uporablja samo za shranjevanje podatkov in dostopanje do njih, temveč je predstavljen kot skupek strežnikov in podatkovnih skladišč, ki predstavljajo zmogljivo strojno opremo. Tako se jih lahko uporabi na zelo različnih področjih. Največkrat so to končni sistemi ali aplikacije, ki za svoje delovanje potrebujejo veliko procesorske moči ali podatkovnega prostora. V tem segmentu je velika priložnost za ustvarjanje novih podjetij oz. novi smeri znotraj njih, ki so opisane v nadaljevanju diplomskega dela. Kot smo že omenili, je dobra stran računalništva v oblaku, da so podatki ves čas dostopni. Prednost pa je tudi ta, da za povezovanje oz. dostop do oblaka ne potrebujemo zmogljive strojne opreme. Zato je lahko uporabljen v pametnih mobilnih napravah, na katerih lahko izvajamo zahtevne algoritme.

V drugem poglavju je na kratko zapisana teorija o računalništvu v oblaku. Predstavljena sta zgodovina in dosedanji razvoj skozi čas. Opisana je tudi arhitektura oblakov in njihove značilnosti. V tretjem poglavju želimo predstaviti nekaj ponudnikov, ki omogočajo uporabo računalništva v oblaku. V njem lahko preberemo o Googlovi, IBM-ovi, Microsoftovi ponudbi in še nekaterih manjših podjetjih s prav tako zanimivo ponudbo. Zanimalo nas je, kaj lahko uporabimo, če se pri njih samo registriramo in tako dobimo brezplačno licenco. Pri vseh pa lahko preberemo, med čim lahko izbiramo ob plačljivi licenci, kakšni viri so nam na voljo in kakšni so stroški ob presežku zakupljenih količin.

Glavni poudarek diplomske naloge je primerjava povezljivosti Android aplikacije s ponudniki v oblaku. V petem poglavju je podrobneje predstavljen način povezovanja z Googlovim oblakom (Google app engine) in Microsoftovim oblakom Azure. Za lažji prikaz smo razvili aplikacijo, ki je namenjena prestrezanju dogodkov, ki jih sprejemamo prek kratkih SMS sporočil. Aplikacija je namenjena shranjevanju podatkov o dogodkih, ki so v našem primeru gasilske intervencije. Zanje je značilno, da se izvedejo na določenem kraju, zaradi česar se je porodila misel, da bi si med potekom same intervencije beležili tudi geografske pozicije.

Zaradi množice podatkov smo si želeli, da bi lahko podatke arhivirali v oblak. V ta namen smo uporabili Google app engine platformo, na katero smo namestili spletno storitev, ki upravlja s shranjevanjem podatkov v tako imenovan *Datastore* (podatkovna baza). Poleg integracije z Googlovim oblakom smo naredili tudi povezavo z Azure platformo. V aplikaciji lahko tako nastavimo, v kateri oblak želimo arhivirati podatke. Čeprav Microsoft skrbi za svoj mobilni operacijski sistem (Windows Mobile), se je izkazal kot zelo odprt tudi za druge razširjene operacijske sisteme (Android in iOS).

Zadnje poglavje je namenjeno sklepni besedi, kjer so predstavljene zamisli in nadgraditve same aplikacije.

Poglavje 2 Računalništvo v oblaku

Računalništvo v oblaku lahko razumemo kot model, ki omogoča omrežni dostop do računalniških virov (strežnik, podatkovna skladišča, aplikacije in storitve). V splošnem lahko računalništvo v oblaku predstavimo kot skupek strežnikov, ki se dinamično prilagajajo glede na potrebe aplikacije. [9] To je ena izmed razlag računalništva v oblaku, ki pojem predstavi najbolj nazorno.

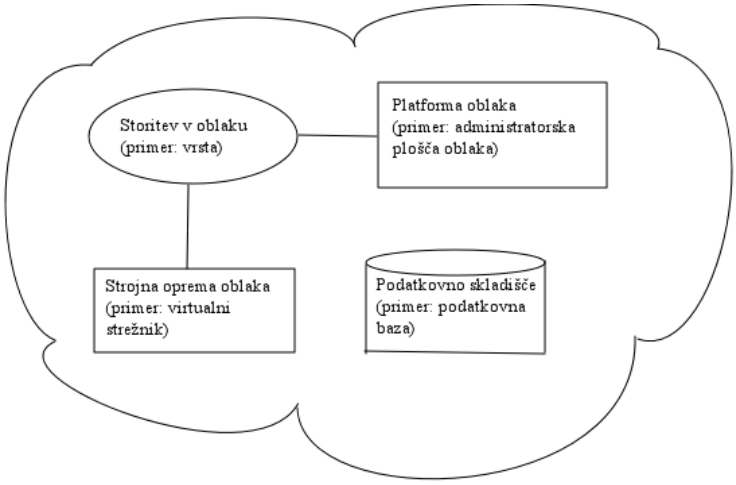
Začetek računalništva v oblaku je predstavljala aplikacija, ki je bila dostopna prek spleta. Sam razvoj le-te pa je omogočila povečana zmogljivost omrežij (hitrost, …). Naslednji mejnik je s spletnimi storitvami, ki so že omogočale shranjevanje in uporabo računalniške moči, postavilo podjetje Amazon. Zatem so sledili še ostali ponudniki, ki so čez nekaj let ponudili podobne storitve. V tem času so se oblikovale tudi skupne značilnosti računalništva v oblaku. V nadaljevanju poglavja pišemo o strukturi in delitvi na tri plasti oblakov. Meja med samimi plastmi je največkrat težko določljiva, saj so med seboj močno povezane.

2.1 Struktura računalništva v oblaku

Računalništvo v oblaku lahko razdelimo na tri plasti. Najvišja plast so aplikacije, srednja je platforma in zadnja, ki je tudi podlaga za preostali dve, je strojna oprema. Na spodnji sliki (slika 1) je razvidna tudi njihova povezanost med seboj. Največkrat so te plasti uporabnikom, bodisi končnim uporabnikom bodisi razvijalcem, ponujene kot storitve. Zato lahko pri tej vrsti računalništva govorimo o storitvenem modelu. Iz tega sledijo pojmi, kot so: aplikacija kot storitev (SaaS), platforma kot storitev(PaaS) in strojna oprema kot storitev (IaaS). Kot že omenjeno, so storitve med seboj povezane, za uspešno delovanje pa je potrebna ustrezna arhitektura.

2.1.1 Arhitektura računalništva v oblaku

Arhitektura računalništva v oblaku je sestavljena iz več delov, ki med seboj komunicirajo. Kot smo že omenili, se do oblaka povezujemo prek storitev, ki skrbijo za komunikacijo s preostalimi deli oblaka. V to storitev se pošiljajo zahtevki, ki se zbirajo v tako



Slika 1: Osnovna arhitektura računalništva v oblaku. Na podlagi: http://en.wikipedia.org/wiki/Cloud_computing

imenovano vrsto (ang. queue). Vrsta je v računalništvu zelo razširjena metoda za obdelovanje zahtevkov, ki so pri računalništvu v oblaku predstavljeni s http sporočili. Metoda jih tako obdeluje glede na vrstni red v vrsti. Najbolj razširjen način obdelovanja se imenuje FIFO, kar pomeni, prvi v vrsto in prvi iz nje (ang. first in, first out).

Kot lahko opazimo na zgornji sliki (slika 1), so storitve le eden izmed štirih delov osnovne strukture oblaka. Če bi želeli vzpostaviti lasten oblak, bi potrebovali primerno strojno in programsko opremo ter podatkovno bazo oz. prostor za shranjevanje podatkov. Slika nam prikazuje tudi, v kakšnem odnosu so posamezni deli. Vsak del je tako povezan prek storitev, ki sprejemajo zahtevke, s čimer se pridobivajo nove naloge, ki jih preostali deli izvršujejo.

Infrastruktura in podatkovna baza nam zagotavljata osnovne vire, ki so uporabljeni za normalno delovanje aplikacij in preostalih dveh storitev. Viri se tako glede na zahtevnost naloge, ki je bila sprejeta prek zahteve, prilagodijo in uporabijo za izvajanje.

2.1.2 Aplikacija kot storitev (SaaS)

Aplikacije kot storitev so dosegljive končnim uporabnikom. Do njih lahko dostopajo z vseh prenosnih naprav. Z uporabniškega vidika so tovrstne aplikacije, ki jih zagotavlja ponudnik, bolj dostopne in uporabne. Najbolj razširjen primer uporabe aplikacije v oblaku so razni odjemalci elektronske pošte, shranjevanje dokumentov itd. V poslovnem svetu pa so te aplikacije namenjene ožji skupini uporabnikov. Primer take aplikacije je upravljanje odnosov s strankami (ang. CRM).[1]

Te aplikacije omogočajo, da so podatki ves čas dostopni. Za to poskrbi ponudnik tako, da ima zagotovljenih dovolj virov. Kot smo že omenili, se viri ves čas prilagajajo, kar predstavlja

Slika 2: Slika prikazuje, kako je sestavljen storitveni model oblaka. Na podlagi: http://en.wikipedia.org/wiki/Cloud_computing

prednost pred klasičnimi namiznimi aplikacijami. Tako se znebimo slabosti namiznih aplikacij, v podjetju pa se zmanjša potreba po zamenjavi strojne opreme.

2.1.3 Platforma kot storitev (PaaS)

Platforma kot storitev v storitvenem modelu računalništva v oblaku predstavlja drugo plast. Uporaba te storitve omogoča dostopnost več uporabnikom, s čimer njena uporaba postane bolj razširjena. Storitev je namenjena predvsem razvijalcem uporabniških aplikacij.

Ekipam razvijalcev je tako olajšan razvoj nove aplikacije. Največkrat je za nov razvoj, poleg ideje, potrebno načrtovati tudi postavitev strežnikov, podatkovne baze, dostopnost do nje, njeno arhiviranje in še marsikaj drugega. V tem primeru to ni potrebno, saj nam to zagotavlja že ponudnik storitve. Ponudniki največkrat ponujajo tudi virtualna razvojna okolja, ki olajšajo sam razvoj. Prek teh okolij razvijalec aplikacijo lahko že testira in opazuje njeno obnašanje v končnem sistemu.

2.1.4 Infrastruktura kot storitev (IaaS)

Prav tako kot v računalništvu je na najnižjem nivoju strojna oprema oz. infrastruktura, ki je potrebna za ustrezno delovanje celotnega sistema. Infrastruktura kot storitev je torej plast v storitvenem modelu, ki zagotavlja vire. Viri so skupek naprav, orodji in mehanizmov, ki skrbijo za usklajeno delovanje sistema. Poleg strežnikov in podatkovnih baz se pod tem

pojmom skrivajo tudi povezovalni in nadzorni mehanizmi, ki skrbijo, da se procesi izvajajo v pravilnem vrstnem redu. Poleg tega med njihove naloge sodi arhiviranje podatkovne baze in ne nazadnje tudi varnost in nadzor nad dostopom do našega okolja. [1]

Storitev nam zagotavlja uporabo virtualnih okolji, operacijskih sistemov in podatkovnega skladišča. Če torej pogledamo izkoriščenost zakupljenih virov, se nam v vsakem primeru izplača uporabljati to storitev. Za preproste aplikacije zakupimo manj virov, s čimer se nam zmanjša strošek, ki ga namenimo infrastrukturi. Pri zahtevnih aplikacijah pa je ta storitev še bolj uporabna, saj se poraba virov prilagaja glede na zahtevnost operacij. V splošnem se v tovrstnih aplikacijah pojavijo ekstremi, ki za primeren odziv potrebujejo veliko virov. Preostali čas pa so viri relativno neizkoriščeni. Ker se stroški izračunajo glede na porabo virov, se na dolgi rok zmanjšajo. Podjetja se z uporabo takšne storitve izognejo nakupu drage fizične strojne opreme in tudi njenemu vzdrževanju.

2.2 Značilnosti

V spodnjih alinejah so predstavljene značilnosti oblakov. Opisane so tako dobre kot tudi slabe lastnosti.

- Zanesljivost: Pojem zanesljivosti v tem primeru predstavlja neprekinjeno dostopnost do storitev oz. aplikacij. Delovanje je brezhibno in tudi če nastanejo težave, jih uporabnik ne opazi. To je zagotovljeno z več kopijami storitve/aplikacije, kar onemogoča izpad storitve, ob nesreči pa je ponovna vzpostavitev delovanja lažja.

- Dostopnost: Kot že sama beseda pove, je to značilnost, ki opredeljuje možnost dostopanja do aplikacij/storitev. Za oblake je znano, da lahko do aplikacij dostopamo iz kjer koli, potrebujemo le internetno povezavo.

- Virtualizacija: To je značilnost, ki predstavlja navidezna okolja, uporabljena na različne načine. Uporabljeni so za testna okolja, s katerimi pridemo do testiranja v končnem okolju. Poleg tega omogoča tudi lažje premeščanje storitev z ene strojne opreme na drugo.

- Stroški na podlagi porabe: Najbolj znan izraz pri računalništvu v oblaku je, da plačaš toliko, kolikor porabiš. Največja prednost je ta, da se poraba virov ves čas spreminja, na podlagi česar se tudi obračunavajo stroški. Ker je izkoriščanje virov v povprečju nizko, je mesečni strošek nižji, kot bi bil pri fizični infrastrukturi. Največkrat obstajajo

tudi brezplačne ponudbe za najem oblaka, presežek teh virov pa se plačuje po ceni, ki je pri vsakem ponudniku nekoliko drugačna.

- Nadzor nad podatki: Z uporabo oblakov lahko dokumente centraliziramo in tako izboljšamo nadzor nad njimi. Ker so zbrani na enem mestu, tudi lažje načrtujemo njihovo varovanje. Pri tej značilnosti lahko omenimo tudi slabost oblakov. Slabost je v tem, da ne vemo natančno, kje se naši podatki nahajajo. Vemo le, da so shranjeni nekje v podatkovnem skladišču. Morda to ni ravno slabost, vendar se pri uporabnikih vzbuja nezaupanje, o čemer se je na začetku razvoja oblakov veliko govorilo. [9]

2.3 Vrste računalništva v oblaku

Uporabniki dostopajo do aplikacij/storitev, ki v ozadju uporabljajo oblak. Tu gre lahko za končne uporabnike, ki spletne aplikacije uporabljajo, lahko pa so to tudi razvijalci, ki potrebujejo infrastrukturo v oblaku. Aplikacije in storitve se v grobem delijo na zasebne (npr. poslovne aplikacije) in splošne oz. javne aplikacije/storitve, ki so dostopne vsem uporabnikom (npr. spletno shranjevanje datotek). Posledica tega so tudi vrste oblakov, ki so razdeljeni glede na dostopnost aplikacije/storitve. Tako jih v grobem delimo na javne, zasebne in hibridne oblake.

2.3.1 Javni oblaki

Javni oblaki so vse storitve, ki so prosto dostopne prek interneta. Do njih lahko dostopajo vsi, ki imajo omrežno povezavo. Pri tem imamo v mislih celoten storitveni model računalništva v oblaku. V poglavju 3 so predstavljeni ponudniki, ki nudijo to vrsto storitev. Nekaj več o sami vrsti je razloženo tudi v omenjenem poglavju.

2.3.2 Zasebni oblaki

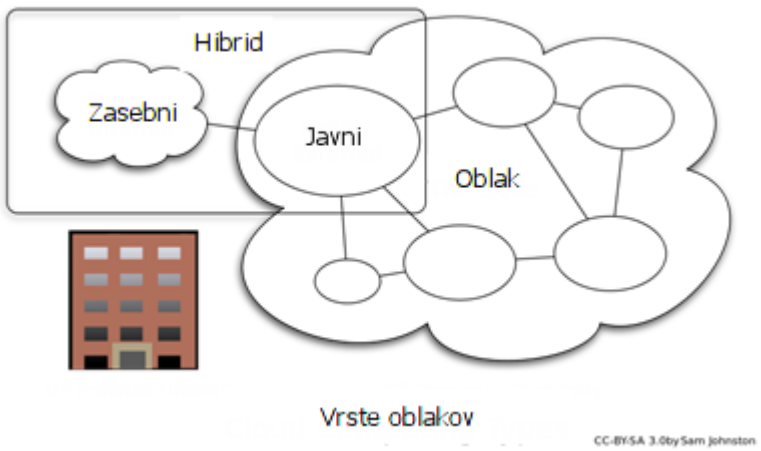
Zasebni oblaki so namenjeni eni organizaciji, pri čemer je lahko strojna oprema oz. infrastruktura v lasti organizacije. V njej so v tem primeru oblikovani oddelki, ki načrtujejo in nadzorujejo njeno delovanje. Lahko pa je to zaupano tudi tretji strani oz. drugi organizaciji, kot so Amazon in podobni ponudniki. Samo načrtovanje postavitve zasebnega oblaka je pod močnim vplivom potreb organizacije. Na podlagi trenutnih kapacitet strojne opreme se izvaja tudi načrtovanje le-te za novonastali oblak. Poleg tega je v ospredju tudi varnost dostopa do aplikacij, storitev in podatkov. Največkrat je to urejeno prek notranje varnostne politike organizacije (domenski uporabniki, uporaba certifikata itd.). Tu opazimo, da mora organizacija poskrbeti za načrtovanje, postavitev in vzdrževanje strojne opreme, zaradi česar

se postavi vprašanje, v čem je prednost vpeljave takega oblaka. Kot prednost lahko izpostavimo lažjo selitev organizacije na drugo fizično lokacijo.

2.3.3 Hibridni oblak

Že samo ime pove, da je to združitev več vrst oblakov. Združuje javni, zasebni in podvrste oblakov. Ena izmed njih se imenuje skupni oblak, ki je pri hibridnih oblakih tudi največkrat uporabljen.

Skupni oblak je podvrsta, ki predstavlja skupne vire več organizacij. To pomeni, da imajo lahko skupno strojno opremo (procesorji, pomnilnik …), kar predstavlja javni del hibridnega oblaka. Zasebni del pa je lahko v tem primeru podatkovno skladišče. Vsaka organizacija načrtuje in skrbi za svojega.[8]



Slika 3: Slika prikazuje vrste oblakov, ki so najbolj razširjene med uporabniki/organizacijami. Na podlagi: http://en.wikipedia.org/wiki/Cloud_computing

Na sliki (slika 3) lahko vidimo primere javnih, zasebnih in hibridnih oblakov. Meje med njimi so določene, vendar pa so tako kot pri storitvah dostikrat težko določljive.

Poglavje 3 Primerjava različnih oblačnih platform

Diplomska naloga je namenjena tudi primerjavi ponudb nekaterih ponudnikov računalništva v oblaku. V prejšnjem poglavju smo opisali teorijo, s katero smo pridobili podlago za lažje razumevanje vsebine tega poglavja. Za podrobno primerjavo smo izbrali šest ponudnikov, ki ponujajo platformo kot storitev. V prvem podpoglavju imamo neposredno tabelarično primerjavo ponudb. O vsaki ponudbi je zapisanih nekaj značilnosti, predstavljeni pa sta brezplačna in plačljiva licenca. Z registracijo aktiviramo brezplačno licenco, s čimer pridobimo enoličen ključ. Ta nam omogoča, da razvijamo aplikacije, ki uporabljajo storitev ponudnika. Če pa navedemo še svoje bančne podatke, pridobimo plačljivo licenco, pri kateri nam ponudnik mesečno obračunava stroške. Za pregled in upravljanje z njimi imamo za to namenjen vmesnik v administratorski plošči oblaka, ki je namenjena tudi nadzoru nad celotno platformo. Prek nje lahko povečamo tudi zmogljivost naših strežnikov, povečamo podatkovni prostor, upravljamo z aplikacijami, pregledujemo napake itd.

3.1 Tabelarična primerjava ponudnikov

Tabelarična primerjava med ponudniki je uporabna, saj dobimo celovit pregled nad vsemi ponudbami. Med seboj primerjamo naslednje ponudnike: Google App engine, Microsoft Azure, CloudBees, Heroku in LongJump. Za primerjavo smo izbrali tiste podatke, ki so bili podani pri večini ponudnikov.

3.1.1 Brezplačna licenca

Kot bomo lahko opazili, vsi ponudniki omogočajo brezplačno licenco, razen IBM, ki za našo državo tega še ne omogoča. Ponudniki, ki so nekoliko bolj neznani, nam ponujajo več, pri čemer izrazito izstopa LongJump. Z brezplačno uporabo dobimo najvišjo moč procesorja (1,2 GHz) in tudi največjo podatkovno bazo (1 TB). Ostale ponudbe pa lahko primerjate v spodnji tabeli (tabela 1).

Ponudniki/ parametri	Google App engine	Azure	CloudBees	Heroku	LongJump
Moč procesorjev (GHz)	600	Prilagodljiva	1,2	Prilagodljiva	1.2 (Amazon)
Velikost pomnilnika (MB)	128	768	128	512	600
Velikost podatkovne baze	1 GB	1 GB	5 MB	10 tisoč vrstic	1 TB
Število uporabnikov	100/dan	/	3	neomejeno	100
Programski jeziki	Java, Python, Go, PHP	PHP, .NET, Java, Python, Ruby	Java, PHP, Ruby	Java, Ruby, Python	Java
Varna povezava (SSL)	Ni omogočena	Ni omogočena	Ni omogočena	Ni omogočena	Ni omogočena

Tabela 1: Primerjava brezplčanih licenc ponudnikov

3.1.2 Plačljiva licenca

Spodnja tabela (tabela 2) prikazuje primerjavo med ponudniki plačljivih licenc. Zopet smo izbrali podatke za primerjavo, ki so bili podani pri večini ponudnikov. Vire pri tej licenci zakupimo vnaprej oz. so vnaprej podani in se potem prilagajajo glede na potrebe aplikacij. Pri večini podatkov je v oklepaju zapisana cena, ki je obračunana, ko prekoračimo zakupljeno vrednost.

	Google AE	Azur	CloudBees	Heroku	LongJump
Moč procesorjev (GHz)	2400 (0,32 $/uro)	1,6 (0,16 $/mes)	6 x 1.2 GHz strežnikov (0,4 $/uro)	Prilagodljiva	Amazon EC2 Od 1.2 naprej (0,02 $/uro)
Velikost pomnilnika (MB)	512	1700	1500	512	600
Velikost podatkovne baze (GB)	1 – zastonj (0,13 $/GB)	5 - zastonj (0,25 $/GB)	5	5	1000
Število uporabnikov	Neomejeno/dan	/	20/dan	/	100/dan
Število storitev	1 GB zastonj (0,13 $/GB)	10 storitev	25 GB	/	50 GB
Število SQL baz	/	20 MB	630MB 0,59 $/uro (Amazon)	Neomejeno	/
Arhiviranje baze	/	5 GB	5 GB	/	/
Programski jeziki	Java, Python, Go,PHP	PHP, .NET, Java, Python, Ruby	Java, PHP, Ruby	Java, Ruby, Python	Java
Varna povezava (SSL)	5 cert./ 9 $	9 $/mes	0,02 $/uro za aplikacijo	20/mes - aplikacija	Omogočena

Tabela 2: Primerjava plačljivih licenc med navedenimi ponudniki

3.2 Google app engine

Podjetje Google je eden izmed ponudnikov, ki ponujajo platformo v oblaku. Njihova platforma se imenuje Google app engine. Na platformo lahko nameščamo lastne aplikacije in spletne strani ter shranjujemo podatke. Prednosti, ki jih poudarjajo, se navezujejo na avtomatično povečevanje potrebnih virov za normalno delovanje aplikacije.

Mesečni stroški se obračunajo glede na porabljene količine, ki so bile potrebne za normalno delovanje aplikacije. Z nastavitvijo finančnega plana (ang. budget) jih lahko omejimo.

3.2.1 Brezplačna licenca

Google ob brezplačni licenci ponuja določene vire, ki jih aplikacija izrablja, ko jo namestimo na njihove strežnike. Kakšni so ti viri, je razvidno iz tabele 1. Če torej zbrane podatke primerjamo z ostalimi, ugotovimo, da so za testiranje aplikacije dovolj, za razširjeno uporabo pa je priporočena uporaba plačljive licence. Moč procesorjev je navzgor omejena (600 GHz), vendar se njihovo število dinamično prilagaja glede na zahteve aplikacije. Podatkovna baza je velika 1 GB, kolikor prostora lahko zasedajo tudi naše aplikacije.

3.2.2 Plačljiva licenca

Če želimo več virov, kot nam jih omogoča brezplačna licenca, moramo omogočiti obračunavanje. Uporaba virov se obračunava mesečno, porabo pa lahko spremljamo v za to namenjenih pogledih znotraj administratorske plošče.

Podatkovna baza se lahko neomejeno razširi. Doplačilo za vsak presežen GB znaša 0,13 dolarja mesečno. V podatkovni bazi lahko nastavimo do 200 indeksov na dan.

3.3 Windows Azure

Windows Azure ponuja več vrst storitev, kot so spletne strani, virtualni strežniki, mobilne aplikacije, podatkovni strežniki itd. Viri za potrebe storitve se avtomatično prilagajajo, stroški pa se računajo na podlagi porabljenih virov. Spreminjanje nastavitev strežnikov (procesorska moč, velikost predpomnilnika itd.) in velikosti podatkovne baze je brezplačno.

3.3.1 Brezplačna licenca

Brezplačna licenca omogoča vzpostavitev novih spletnih in mobilnih aplikacij. Ostala področja niso vključena, saj je zanje potrebno aktivirati plačljivo licenco.

Objavimo lahko do 10 spletnih aplikacij, podatke iz njih pa se lahko shrani v 1 GB veliko relacijsko podatkovno bazo.

Mobilna storitev je predpripravljena za razvoj aplikacij v Windows Mobile, Apple iOS in Android operacijskih sistemih. Prek nje lahko shranjujemo podatke v bazo, poleg tega pa omogoča tudi preverjanje identitete uporabnika in neposredno sporočanje v realnem času. Razvijemo in ponudimo lahko do 10 aplikacij mesečno. V celotnem času aplikacij je mogoče izvesti 500.000 dostopov, do aplikacije pa lahko dostopa 500 različnih naprav.

3.3.2 Plačljiva licenca

V plačljivi licenci lahko poljubno določimo moč strojne opreme. Poleg spletnih strani in mobilnih aplikacij pa sta omogočeni tudi storitev za shranjevanje v podatkovna skladišča ter uporaba virtualnih računalnikov.

Na strojni opremi lahko poganjamo največ 20 spletnih aplikacij, ki bi mesečno stale 190 dolarjev.

Relacijska SQL podatkovna baza se lahko razširi do 150 GB prostora, za kar bi mesečno plačali 225 dolarjev. Prenos podatkov se zaračunava samo, ko se podatki pridobivajo iz baze,

medtem ko vhodni podatki niso obračunani. Pri tem velja, da več kot prenesemo, nižja je cena za en preneseni GB.

Mobilna storitev, ki je vključena v brezplačno licenco, se razdeli v dve plačljivi ponudbi. Prva se imenuje »Standard«, druga pa »Premium«. Standard ponudba za eno aplikacijo, ki stane 25 dolarjev na mesec, vključuje 1,5 milijonov klicev aplikacije in neomejeno število naprav, ki lahko dostopajo do njih. Premium izbira medtem ponuja 15 milijonov klicev naših aplikacij pri neomejenem številu naprav, ki lahko dostopajo do njih. Mesečna cena za najem takšne storitve je 199 dolarjev.

Velikost SQL baze je na začetku 20 MB, vendar jo lahko povečamo vse do velikosti 150 GB. Kot že omenjeno, so plačljivi samo odhodni podatki, največja velikost izhodnih podatkov na mesec pa je 2 GB. Za takšno možnost bi mesečno doplačali 240 dolarjev.

Kot pri vseh ponudnikih se v oblaku shranjujejo tudi podatki. Pri tej licenci lahko izbiramo med različnimi razporeditvami podatkovnih baz, kar pomeni, da so lahko vse na eni lokaciji ali pa so krajevno neodvisne. Zadnja je zaradi tega nekoliko dražja, vendar je tudi varnejša. Prvi TB za fizično ločene podatkovne baze stane 0,095 dolarja, za ostale pa 0,07 dolarja.

3.4 IBM

Pri podjetju IBM so razvili storitev poimenovano »IBM SmartCloud«, ki omogoča, da svoje dosedanje poslovne procese preselimo na prilagodljivo infrastrukturo. S tem dosežemo boljšo učinkovitost in spodbudimo inovativne zamisli.

Oblak je razdeljen na module, na podlagi katerih je določeno, kateri viri in dodatki so podprti. Prvi modul je namenjen medsebojnemu pošiljanju elektronske pošte in urejanju skupnega koledarja. Naslednji je namenjen predvsem komunikaciji, kar vključuje neposredno sporočanje med člani skupine, urejanje svojih kontaktov, dodajanje datotek v skupno rabo itd. Tretji modul pa omogoča spletne konference in dogodke.

3.4.1 Brezplačna licenca

Trenutno preizkusna verzija, ki traja 60 dni in se nato nadgradi na plačljivo, v Sloveniji še ni na voljo. V testni različici so zagotovljeni štirje (virtualni) strežniki. Podprta sta 32- in 64-bitni operacijski sistem. Lahko izbiramo med operacijskimi sistemi Windows, Linux in Linuxovimi podvrstami. Na tej infrastrukturi lahko teče pet različnih aplikacij, zagotovljena velikost podatkovne baze pa znaša 60 GB.

3.4.2 Plačljiva verzija

V plačljivi verziji lahko izbiramo med desetimi strežniki, ki jih lahko opremimo z različnimi operacijskimi sistemi in določimo njihov namen. Kot že zgoraj omenjeno, lahko izbiramo med strežnikoma 2008 in 2012 operacijskega sistema Windows, pri čemer sta pri obeh različicah na voljo 32- in 64-bitni sistem. Te strežnike lahko potem še množimo do poljubnih 10.000 instanc strežnika. Odločamo tudi o tem, koliko procesorjev naj ima ena enota in kako velik naj bo pomnilnik. Pri 32-bitnih sistemih imamo lahko 1, 2 ali 4 procesorje in 4 oz. 8 GB pomnilnika. Pri 64-bitnih sistemih pa imamo lahko v eni enoti največ 16 procesorjev in do 16 GB pomnilnika. Moč procesorja znaša 1,25 GHz, v odvisnosti od tega pa nam pripadajo tudi različno velike podatkovne baze. Njihova velikost se začne pri 60 GB, konča pa 350 GB (32 bitov) oz. 2048 GB (64 bitov). Cena najslabše ponudbe se tako začne pri 80 dolarjih na mesec, najmočnejša izbira pa bi nas stala več kot 1.500 dolarjev mesečno.

3.5 CloudBees

CloudBees je ponudnik oblaka, ki ponuja platformo kot storitev. Platforma je razvita v programskem jeziku Java. Platforma tako podpira vse aplikacije, razvite v programskih jezikih, ki se jih lahko poganja v Java virtualnem strežniku (JVM -Java virtual machine). To so jeziki Java, Spring, JRuby, Grails in Scal. CloudBees nima lastne strojne opreme, zaradi česar ponuja strežnike različnih proizvajalcev, ki jih ob registraciji izberemo. Izbiramo lahko med HP-jevimi in Amazonovimi strežniki.

3.5.1 Brezplačna licenca

Pri tem ponudniku je posebnost ta, da je storitev pri tej licenci namenjena samo razvijalcem. V brezplačni različici imamo na voljo najšibkejši procesor, ki se nahaja na Amazonovem strežniku (600 GHz in 1 GB pomnilnika). Do aplikacije lahko dostopajo največ trije razvijalci, velikost aplikacije pa ne sme presegati 2 GB prostora. Poleg tega lahko pri podatkovni bazi izbiramo med MongoHQ-jevimi in Amazonovimi strežniki. Ostale podatke lahko preberemo v zgornji tabeli (tabela 1).

3.5.2 Plačljiva licenca

Pri razvojnem oblaku pa lahko zakupimo tudi določene druge vire. Izbiramo lahko med moduloma »PRO« in »ENTERPRISE«, ki omogočata šibkejši (0.106 dolarja/uro), kot tudi močnejši (0.425 dolarja/uro) strežnik. Istočasno lahko teče več aplikacij. Pri modulu PRO lahko vse storitve in aplikacije zasedejo 25 GB, medtem ko pri modulu ENTERPRISE lahko

zasedejo do 50 GB. Do aplikacije lahko dostopa 20 (PRO) oz. 50 (ENTERPRISE) uporabnikov.

Shranjevanje podatkov na strežnikih je omogočeno tudi pri tej licenci. Poleg brezplačne različice sta na voljo še dve ponudbi. Prva vključuje 5 GB veliko podatkovno bazo, na katero se lahko povezuje največ 20 aplikacij. Pri njej je strojna oprema vnaprej določena (1,2 GHz procesorske moči in 1 GB pomnilnika). Arhiviranje baze se izvaja dnevno, arhivi pa so dosegljivi za pet dni nazaj. Cena tovrstne ponudbe je 25 dolarjev mesečno. Druga možnost, ki omogoča več proste izbire, pa cene nima natančno določene. Odvisna je od tega, kako močno strojno opremo izberemo (število strežnikov, ki ima 1,2 GHz močan procesor in 1GB pomnilnika). Cena se giblje od 150 do 650 dolarjev. Omejeni smo samo z velikostjo podatkovne baze, ki ne sme presegati 50 GB.

3.6 Heroku

Heroku je ime produkta, ki ga ponuja podjetje SalesForce. To je podjetje, ki ponuja izključno rešitve povezane z oblakom. Heroku je produkt, ki na ponudnikovi platformi omogoča podporo aplikacijam.

SalesForce ponuja predvsem rešitve v poslovnem svetu, saj ponujajo sisteme za pregled poslovanja podjetja, ki je namenjeno predvsem direktorjem in pa tudi tržnikom, ki imajo npr. dostop do sistema za upravljanje z uporabniki (CRM).

Heroku je usmerjen k temu, da razvijalec lahko izbere med veliko dodatnimi stvarmi. To so na primer, katera podatkovna baza se bo uporabljala (SQL, MySQL, Oracle …), kakšne bodo medijske podpore v aplikaciji, posebni dodatki za beleženje akcij v aplikaciji itd. Brezplačna ponudba sestavlja osnovne vire za razvoj in testiranje aplikacije, plačljivo pa lahko poljubno prilagajamo. Heroku je znan kot zelo agilen in prilagodljiv oblak.

3.6.1 Brezplačna licenca

Moč strežnikov določamo prek števila tako imenovanih »dynos«, ki jih lahko imenujemo tudi enote. Pri brezplačni ponudbi je zagotovljena ena enota s 512 MB pomnilnika in enim procesorjem (600 GHz), ki se prilagaja potrebam aplikacije. Podatkovna baza, v katero lahko zapišemo do 10.000 zapisov, temelji na PostgerSQL. Ta ponudba je namenjena predvsem razvijalcem aplikacij.

3.6.2 Plačljiva licenca

Pri tej ponudbi se osnovna ponudba moči enot in strežnika za podatkovno bazo poveča. Pri enotah lahko izbiramo med dvema stopnjama moči. Prva možnost zagotavlja 512 MB pomnilnika in en procesor, druga pa 1024 MB pomnilnika in dva procesorja. Hitrost procesorja je prilagodljiva glede na potrebe aplikacije. Pri eni in drugi možnosti lahko zakupimo do 150 enot. Šibkejša izbira stane 0,05 dolarja na uro za enoto, druga cena pa je dvakrat višja, in sicer 0,10 dolarja na uro za enoto.

Ponudba za podatkovno bazo je velika, saj lahko izbiramo med kar 12 različnimi strežniki. Velikost podatkovne baze lahko doseže 1 TB, če izberemo najmočnejše strežnike. Mesečna cena se nahaja v razponu od 50 do 6000 dolarjev.

3.7 LongJump

LongJump je produkt, ki deluje na več področjih (telekomunikacije, zdravstvo, zelena energija …) in pri katerem vse storitve tečejo v oblaku. Ponujajo tudi oblak kot platformo, na katero lahko, tako kot povsod drugod, nameščamo svoje aplikacije. Aplikacije morajo biti razvite v programskem jeziku Java.

Produkt ponuja podjetje Software AG, ki deluje na področjih poslovnih procesov, novih integracij s sistemi in rudarjenja nad podatki. Pri svojih ostalih produktih največkrat uporabljajo oblak.

Ta izdelek združuje več področij, na katerih deluje ponudnik. Združuje namreč oblak, mobilne storitve in obdelavo velike količine podatkov.

3.7.1 Licenca

Pri tem ponudniku se plačljiva in brezplačna ponudba ne razlikujeta. Prvih 14 dni je licenca, ki naj se bi uporabljala za razvoj in nameščanje aplikacij na ponudnikove strežnike, brezplačna. Po pretečeni brezplačni licenci je potrebno za podaljšanje licence plačati 49 dolarjev mesečno. Na platformo lahko naložimo do 100 aplikacij, prostora zanje pa je 50 GB. Ko to mejo presežemo, moramo za 10 GB doplačati 50 dolarjev mesečno.

Strežniki, na katerih stoji platforma, je lahko v lasti kupca, lahko pa najamemo strežnike, ki jih ponujajo drugi večji ponudniki. Tako lahko poljubno izbiramo moči procesorjev in pomnilnikov ter ostalo strojno opremo. Največkrat se pojavijo v povezavi z LongJumpovimi strežniki, ki so v Amazonovi lasti. Ta ponuja strežnike za podporo oblakom pod oznako EC2. Pri njem imamo velik nabor ponudb. Večina strežnikov ima nameščen operacijski sistem

Linux, lahko pa namestimo tudi Windows OS. Za najcenejši strežnik, ki ima 600 MB pomnilnika in 1.2 GHz močan procesor, bi mesečno plačevali 15 dolarjev. Najzmogljivejši strežnik premore 16 procesorjev in 117 GB pomnilnika, mesečno plačilo za to ponudbo pa znaša 4.600 dolarjev.

Poglavje 4 Uporabljene tehnologije in programi

Za razvoj programskih rešitev potrebujemo različna orodja, ki uporabljajo tehnologije in vtičnike za lažji razvoj. Za potrebe primerjave med ponudniki smo najprej razvili Android aplikacijo. Android je odprtokodni operacijski sistem, za katerega lahko razvijamo različne aplikacije v programskem jeziku Java. V ta namen se najpogosteje uporablja program Eclipse, ki je v nadaljevanju na kratko opisan. Opisana sta tudi Android vtičnik (SDK) in virtualna mobilna naprava (ang. Emulator).

4.1 Program Eclipse

Gre za odprtokodno orodje, ki se je začelo razvijati v letu 2001. Sam razvoj in uporabo je najbolj podpiralo podjetje IBM, sčasoma pa se je ta krog podpornikov povečeval (Google, Oracle …). V njem je možno razvijati aplikacije, ki so razvite s programskimi jeziki C, C++, Java …, omogoča pa tudi lokalno poganjanje aplikacij in njihovo testiranje. Poleg zgoraj naštetih programskih jezikov omogoča uvažanje dodatnih knjižnic in vtičnikov, s čimer lahko pospešimo razvoj aplikacije, saj uporabljamo že razvite rešitve. [4]

4.2 Operacijski sistem Android

Operacijski sistem Android [5] je sistem, ki so ga začeli razvijati leta 2003. Takrat je podjetje Google, ki je tudi najbolj zaslužno za tak napredek, ustanovilo združenje, ki se je zavzemalo za napredek na področju pametne telefonije in ostalih prenosnih naprav. Prva verzija (1.0) je izšla leta 2007.

Odlika tega operacijskega sistema je odprtokodni sistem, ki omogoča lažji in cenejši razvoj aplikacij. Dobra stran je tudi ta, da je sistem večopravilen, odziven in enostaven za uporabo.

4.3 Android SDK

Android SDK [6] je skupek orodji, ki omogočajo razvoj in testiranje Android aplikacij. Uvozi se jih v razvojni program, ki je v našem primeru Eclipse. Sestavljajo ga med seboj združljive API (angl. application programming interface) knjižnice, ki podpirajo novosti pri posamičnih

verzijah operacijskega sistema Android. Z virtualnimi mobilnimi napravami (ang. Android emulator) nam Android SDK pomaga pri testiranju razvitih aplikacij. Poleg testiranja nam omogoča tudi prenos na lastno napravo, s čimer lahko razvite aplikacije preizkusimo v realnosti.

4.4 Android Emulator

Emulator [7] je virtualna mobilna naprava, ki deluje na našem lokalnem računalniku. S pomočjo tega orodja lažje simuliramo obnašanje aplikacije v realni uporabi, ocenimo lahko ustreznost uporabniškega vmesnika, prehode med samimi aktivnostmi ter prikazom opozoril in slik itd. Emulator nam tako omogoča testiranje na različnih napravah in tudi na različnih verzijah Android OS. Z njegovo uporabo lahko nadziramo in opazujemo dogajanje v ozadju aplikacije, pri čemer mislimo na dostop do lokalne podatkovne baze, ki jo uporablja aplikacija. Poleg tega lahko dostopamo tudi do datotek, v katere aplikacija zapisuje podatke.

Poglavje 5 Primerjava povezovanja aplikacije v OS Android z različnimi oblačnimi platformami

Namen tega poglavja je, da s praktičnimi primeri predstavimo povezljivost s ponudniki, ki smo jih opisali v prejšnjih poglavjih. Najprej smo razvili aplikacijo, ki smo jo kasneje uporabili za testiranje oz. prikaz povezovanja s platformami v oblaku.

Uporabniško aplikacijo smo razvili v programskem jeziku Java za operacijski sistem Android. Uporabniku ponuja pregledovanje posameznih dogodkov, ki so v našem primeru gasilske intervencije. Nov dogodek se ustvari, ko prejmemo kratko sporočilo (SMS), vsi podatki o dogodkih pa so shranjeni v lokalni SQLite podatkovni bazi.

Podatke, ki so shranjeni v bazi, je potrebno arhivirati. Zato smo se odločili, da bomo podatke pošiljali v podatkovne strukture ponudnikove platforme v oblaku. Vsi ponudniki podpirajo povezovanje z Android aplikacijami, težava pa nastane, ko je potrebno vzpostaviti povezavo med aplikacijo in storitvijo ponudnika. Kako smo to rešili oz. kaj ponujajo ponudniki za lažjo povezavo, lahko preberete v tretjem in četrtem podpoglavju.

Predstavitev same aplikacije smo razdelili na dva dela, in sicer na čelni in zaledni sistem. Ker pa je glavna tema diplomske naloge prikaz uporabe storitev v oblaku z Android aplikacijo, je opis aplikacije nekoliko okrnjen.

5.1 Razvoj čelnega sistema

S čelnim delom uporabnik komunicira prek uporabniškega vmesnika, njegovo glavno okno pa je predstavljeno s seznamom dogodkov, ki smo jih prejeli. Dogodki se ustvarijo ob prejemu kratkih SMS sporočil.

5.1.1 Grafični uporabniški vmesnik

Operacijski sistem omogoča uporabo mnogih grafičnih komponent, s katerimi gradimo uporabniški vmesnik aplikacije. Z velikim naborom komponent lahko ustvarimo poljubno oblikovane aplikacije. Osnova za postavitev gradnikov je razporeditev na ekranu (ang. Layout). Uporabljene so preproste komponente, kot so prikaz besedila, vnos besedila itd.

Omogoča pa tudi bolj kompleksne tipe, kot so prikaz slik in filmov ter animacija. S takšnim naborom komponent lahko ustvarimo poljubno oblikovane aplikacije, ki so uporabniku prijetne za uporabo. [2] [10]

Uporabniški vmesnik je umeščen v okno, kar je v ozadju aplikacije predstavljeno kot aktivnost. Aktivnosti, ki jih je lahko poljubno mnogo in so v ozadju povezane, so glavni del aplikacije. V celotni aplikaciji je ena aktivnost glavna, to je največkrat tista, ki se prikaže kot prvo okno. Glavna aktivnost poda uporabniku osnovne podatke za nadaljnje delo oz. vsebuje navigacijski meni.

V splošnem aplikacije operirajo s podatki, ki jih je potrebno shraniti. Potrebno je shraniti že podatke o sami aktivnosti (kaj prikazuje, kateri gumb smo pritisnili …). Nekatere shranjujejo tudi večje količine podatkov, zaradi česar se uporablja shranjevanje v datoteke ali pa v podatkovno bazo. Za našo aplikacijo smo izbrali slednje. [11]

Kot že omenjeno, smo čelni del razdelili na več aktivnosti, in sicer:

- Pregled dogodkov: prikazan je seznam vseh dosedanjih dogodkov. Urejeni so po padajočem vrstnem redu.
- Detajlno okno: prikazuje podrobnosti o dogodku. Prikaže se besedilo, ki smo ga prejeli prek sporočila. Prikažejo se tudi datum začetka, datum konca in trajanje dogodka, če je že zaključen.
- Nastavitve: aktivnost za spreminjanje nastavitev aplikacije.

5.1.1.1 Pregled dogodkov

Ob pritisku na ikono aplikacije v glavnem meniju operacijskega sistema se nam odpre aktivnost pregled dogodkov. To je v našem primeru glavna aktivnost aplikacije.

Kot lahko vidimo na sliki (slika 4), so dogodki urejeni v seznam. Prikazani so vsi dosedanji dogodki, ki so se že odvili, in tudi tisti, ki so v teku. Urejeni so po padajočem časovnem vrstnem redu. Dogodki so na začetku v izvajanju, zaradi česar smo kot indikator dodali zelen trak. Ko se dogodek zaključi, se indikator odstrani, s čimer dosežemo tudi boljšo preglednost nad samimi dogodki.

Slika 4: Prikaz glavnega okna aplikacije

5.1.1.2 Detajlno okno

Aktivnost se prikaže, ko izberemo enega izmed dogodkov na seznamu. V ozadju operacijski sistem zazna dotik in s tem pozicijo. Na podlagi tega je izbran tisti dogodek, ki smo ga želeli. Ob tem se v aplikaciji sproži metoda, ki poskrbi, da se podatki iz podatkovne baze prenesejo v novo okno. Prikaže se nov uporabniški vmesnik (slika 5).

5.1.1.3 Nastavitve

Do te aktivnosti lahko dostopamo prek gumba za nastavitve, ki je določen s strani operacijskega sistema. V nastavitvah lahko nastavimo napis na vrhu zaslona, ki je v našem primeru intervencija. Spremenimo lahko tudi, s katere številke prejemamo SMS sporočila, ki so vir za nastanek novega dogodka. Pošiljanje podatkov v oblak omogočimo s kljukico »pošlji v oblak«. Poleg tega imamo tudi možnost izbire med ponudniki, s katerimi je aplikacija povezana. Vključili smo možnost izbiranja s spustnega seznama, s čimer izberemo, kam se bodo dogodki arhivirali.

Slika 5: Uporabniški vmesnik za aktivnost »Detajlno okno«

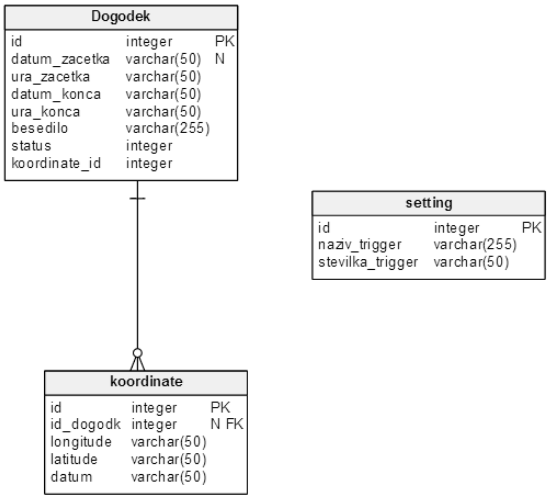
5.2 Razvoj zalednega sistema

Kot razkriva že naslov, bo v naslednjih odstavkih govora o zalednem sistemu naše aplikacije. V grobem ima aplikacija v ozadju dve pomembni stvari, to sta podatkovna baza in tako imenovani poslušalec (ang. Listener) za prispela SMS sporočila.

Uporabljena je relacijska podatkovna baza SQLite, ki je zelo preprosta. Sestavljata jo dve povezani relaciji in ena nepovezana, ki služi nastavitvam aplikacije. Poslušalec za SMS sporočila je namenjen temu, da se, ko prispe novo sporočilo ob izpolnitvi pogoja, v naši aplikaciji ustvari nov dogodek. Ta aktivnost v ozadju ves čas teče in preži nad novimi prispelimi sporočili.

5.2.1 Podatkovna baza

Tipična podatkovna baza v operacijskem sistemu Android je relacijska SQLite podatkovna baza. Značilnost te vrste podatkovne baze je v tem, da za svoje delovanje ne potrebuje posebnega sistemskega procesa. Sama podatkovna baza je predstavljena s podatkovno datoteko, ki jo lahko najdemo na primarnem disku operacijskega sistema.



Slika 6: Konceptualni model podatkovne baze

Kot smo omenili že v uvodu poglavja, naša podatkovna baza vsebuje tri relacije, ki skrbijo za nemoteno delovanje aplikacije. Osrednja relacija je tabela »Dogodek«, v kateri so shranjeni podatki o vseh dogodkih. Kot lahko vidimo iz slike (Slika 6), v njej shranimo besedilo dogodka, začetni in končni datum ter status (s tem označimo, ali je dogodek še v teku ali ne). Ko se dogodek ustvari, se v ozadju sproži tudi metoda za beleženje koordinat. Vse se zabeležijo v tako imenovano tabelo, ki beleži geografsko širino in dolžino ter datum vpisa. Primarni ključ pri obeh tabelah je poimenovan z »id«, ob dodajanju novih dogodkov pa se avtomatsko povečuje.

5.2.2 Poslušalec prejetih kratkih sporočil

Poslušalec (ang. Listener) je splošen programski koncept v Android SDK . Kadar se zgodi neka akcija (pritisk na gumb, dostop do menija z nastavitvami itd.) se sproži nek dogodek (ang. event), ki ga v ozadju prestreže poslušalec.

Prestrezanje kratkih besedilnih sporočil omogočimo z razredom `BroadcastReceiver` [12]. To je razred, ki se ga uporablja pri komunikaciji s preostalimi dogodki v sistemu. Tipičen primer uporabe je skoraj prazna baterija, takrat v ozadju poskrbi za izvedbo določene procedure. Koda je zapisana v metodi onRecieve, ki je obvezna metoda in jo je potrebno prepisati (ang. override). Metoda se zažene tudi ob inicializaciji tega razreda.

S pravicami moramo omogočiti nadzor nad novo prispelimi sporočili. Tako moramo v glavno nastavitveno datoteko (največkrat manifest.xml) aplikacije dodati naslednji dve pravici:

- `android.permission.READ_SMS`,
- `android.permission.RECEIVE_SMS`.

Kot že omenjeno, je metoda onRecieve obvezna metoda, kako aplikacija obravnava sporočila pa lahko vidimo v spodnji kodi. (koda 1)

```
public class SmsListener extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
      if(intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED")){

     //pridobimo besedeilo spročila

      Database db = new Database(context);
      String id = db.insertValue(msgBody);
      db.close();

      if(Setting.getVOblak()>0){
            if(Setting.getOblakPonudnik() == 0)
                  MainActivity.addInter(id, msgBody);
            else
                  MainActivity.addInterAzure(id, msgBody);

       //prikaže se detajlno okno
       }
```

Koda 1: Prikaz kode, ki implementira poslušalca za kratka SMS sporočila

Preden se prikaže novo okno, se podatki zapišejo v podatkovno bazo. Če imamo omogočen zapis v oblak, se v tem trenutku zgodi tudi vpis v podatkovne strukture oblaka prek storitve. Ob uspešnem zapisu se zažene tudi metoda za pridobivanje koordinat.

5.3 Povezava s ponudnikom oblačnih storitev – Google app engine

V tem poglavju opisujemo povezavo med Android aplikacijo in Google app engine (v nadaljevanju GAE). Na kratko je opisano ustvarjanje aplikacije v GAE. Da smo lahko povezali aplikacijo z oblakom smo razvili vmesnik, ki predstavlja RESTful aplikacijo. Ta prek storitev omogoča pošiljanje podatkov iz aplikacije v podatkovno bazo v oblaku.

Razlog za izbiro GAE je povezanost med Androidom in podjetjem Google. Kot smo pričakovali, je povezava med aplikacijo in oblačno platformo dobro urejena (vtičniki za program Eclipse, veliko literature zgledov za povezovanje itd.). Za našo aplikacijo je bilo pomembno, da ima ponudnik možnost shranjevanja podatkov v podatkovno bazo.

5.3.1 Google app engine z uporabniškega vidika

Kot vsak ponudnik platform v oblaku od nas zahteva registracijo v sistem, kar nam omogoči uporabo njihovih funkcionalnosti. Po potrditvi prijave lahko že ustvarimo prvo aplikacijo. Vse, kar je potrebno narediti je, da vpišemo poljubno ime, ki ne sme vsebovati presledkov in tako se naša platforma v oblaku imenuje »fire-for-alert«. Ime bo kasneje uporabljeno za povezavo iz Android aplikacije. Po uspešni registraciji lahko dostopamo do nadzorne plošče naše platforme.

Razvijalci storitev imajo možnost namestitve vtičnika (Google app engine SDK), ki je pripravljen za delo v programu Eclipse. S tem si precej olajšamo razvoj, saj nam ta vtičnik omogoča vse potrebno za uporabo metod za povezovanje s platformo. Dobra stran SDK-ja pa je tudi ta, da lahko neposredno razvito storitev naložimo v oblak in je tako že pripravljena za širšo uporabo.

5.3.2 Storitev za delo z oblakom

Kot smo omenili v uvodu tega poglavja, se po registraciji ustvari dostop do platforme, na katero lahko dodajamo aplikacije/storitve, ki dostopajo do virov. Te aplikacije predstavljajo nekakšen vmesnik za delo z oblakom. V našem primeru je to storitev, ki upravlja s podatkovno bazo. Za razvoj naše aplikacije smo uporabili Restlet odprtokodni vmesnik, ki je del REST arhitekture.

Restlet vmesnik se je pojavil leta 2005 in je odprtokodni API za Java aplikacije. Uporaben je tako v uporabniških aplikacijah kot na strežniških storitvah. Prav tako omogoča vse osnovne oblike podatkov, internetne komunikacije in tudi metode za prenos informacij v spletu (HTTP/S, XML, JSON, …). [3]

5.3.2.1 Strežniška aplikacija - storitev

Naša strežniška storitev je nameščena v oblaku in se uporablja za zapis v Googlovo bazo *Datastore.* Baza lahko vsebuje več relacij. Vsaka ima lahko več stolpcev, zapis pa je predstavljen z enolično številko, ki jo relacija ob novem vnosu dodeli avtomatsko. V našem primeru bo podatkovna baza vsebovala relaciji Inter (podatki o dogodkih) in Coordinate (podatki o položaju).

Za ustvarjanje storitve, ki je povezana z GAE, moramo dodati naslov, prek katerega dostopamo do naše platforme v oblaku. S tem omogočimo uporabo virov in vseh vtičnikov, ki so omogočeni. Aplikacija je izpeljana iz knjižnice *Restlet Aplication*, ki ustvari navidezni vhod (ang. virtual host), prek katerega poteka vsa komunikacija z odjemalci, ki pošiljajo zahtevke (ang. request). Velikokrat imamo več različnih zahtevkov, ki se obravnavajo na

različne načine. Za to poskrbimo tako, da dodamo več navideznih pod-vhodov. Kot lahko vidimo, za vsak pod-vhod določimo razrede, ki obravnavajo prejete zahtevke (koda 2). Predstavljeni so kot kontrolorji, ki pridobljene podatke usmerjajo v pravo relacijo. [3]

```
public class RestletDispatch extends Application
{
	@Override
	public synchronized Restlet createInboundRoot()
	{
		final Router router = new Router(getContext());

		router.attach("/coordinate", CoordinateController.class);
		router.attach("/inter", InterController.class);

		return router;
	}
}
```

Koda 2: Prikaz deklaracije Restlet aplikacije

GAE za delo s podatkovno bazo ponuja dve knjižnici, in sicer Objectify [13] ter Slim3. V našem primeru smo izbrali prvo, ki je namenjena preprostejšim zahtevkom (preproste poizvedbe), saj ponuja osnovne operacije za pridobivanje, vpis in izbris podatkov.

V kontrolnem razredu najprej registriramo relacijo (ukaz *register*), kjer podamo razred, ki jo določa (primer koda 3). Ta je določena z razredom, kjer so opredeljeni tudi atributi oz. stolpci v njej. Če relacija s tako strukturo še ne obstaja, se ustvari nova. Z ukazom *begin* odjemalec vzpostavi novo povezavo, prek katere pošilja zahtevke. Podatki so poslani prek njih in se zapisujejo v relacijo.

```
public void create(Coordinate coordinate) {
ObjectifyService.register(Coordinate.class);

	Objectify ofy = ObjectifyService.begin();

	Coordinate tp = new Coordinate();
	tp.setIdIntervencije(coordinate.getIdIntervencije());
	tp.setLatitude(coordinate.getLatitude());
	tp.setLongitude(coordinate.getLongitude());
	tp.setDatum(coordinate.getDatum());
	ofy.put(tp);
}
```

Koda 3: Prikaz uporabe knjižnice Objectify in razreda »Coordinate«

5.3.2.2 Uporabniška aplikacija – odjemalec

Odjemalec je v našem primeru aplikacija, ki teče na operacijskem sistemu Android. Kot smo že omenili, bi aplikacija uporabljala zgoraj opisano storitev za arhiviranje podatkov o

dogodkih in tudi o samih pozicijah med njihovim izvajanjem. V aplikaciji je prav tako definiran vmesnik, ki vzpostavi povezavo s storitvijo, pripravili pa smo tudi razred za pošiljanje podatkov.

V naši Android aplikaciji smo ustvarili nov razred, ki vzpostavi povezavo s storitvijo v oblaku. V konstruktorju razreda z atributi določimo, katere podatke bomo poslali. Restlet knjižnica največkrat uporablja obliko zapisa ključ – vrednost (XML ali JSON format). V ta namen je v knjižnico že vključen pretvornik `JacksonConverter`, ki podatke preoblikuje tako, da jih storitve lažje obdelujejo. Poleg pretvornika je potrebno določiti tudi protokol za pošiljanje zahtevkov. V našem primeru smo izbrali preprost http protokol in tako definirali tudi pot do naše spletne storitve (koda 4).

```
public final static String gae_path = "http://fire-for-alert.appspot.com";

public EngineConfiguration()
{
       Engine.getInstance().getRegisteredConverters().add(new JacksonConverter());
       Engine.getInstance().getRegisteredClients().add(new HttpClientHelper(null));
}
```

Koda 4: Primer potrebnih nastavitev za povezovanje s spletno storitvijo v GAE

Tako kot pri spletni storitvi je potrebno tudi tu za vsako relacijo ustvariti posebnega upravljavca, ki pošilja http zahtevke v našo storitev. Kot smo omenili, ima spletna storitev tudi podvhode, na podlagi katerih razlikujemo zahtevke in njihovo obravnavo. Podvhode v samem upravljavcu določimo tako, da poleg naslova do storitve podamo še naslov vhoda (primer v kodi 7). Ker naša aplikacija predstavlja odjemalca, se v ta namen uporabi `ClientResource` razred, ki teče na strani uporabniške aplikacije. Vsakega upravljavca ovijemo (ang. wrap) z razredom, ki predstavlja relacijo v podatkovni bazi oblaka. S to ovojnico določimo tudi metode, ki skrbijo za pošiljanje zahtevkov in sprejem odgovorov iz storitve v oblaku (koda 5).

```
public ClientResource cr = new ClientResource(EngineConfiguration.gae_path +
"/rest/inter");

public void create(Inter inter) throws Exception {
final InterControllerInterface uci =
      cr.wrap(InterControllerInterface.class);
try {
          uci.create(inter);
          Log.i("InterController", "Creation success !");
      }
      catch (Exception e) {
          Log.i("InterController", "Creation failed !");
          throw e;
      }
}
```

Koda 5: Primer kode za vzpostavitev povezave in pošiljanje podatkov v spletno storitev

5.3.3 Uporaba spletne storitve

V prejšnjem poglavju smo spletno storitev in odjemalca opisali bolj tehnično. Iz podanih delov kode lahko razberemo, da je za uspešno vzpostavitev povezave potrebnih nekaj nastavitev. Sama uporaba te povezave je bila za našo aplikacijo precej enostavna, saj smo v glavni aktivnosti aplikacije zapisali metodi, ki jih uporabimo na mestih, kjer se sproži zapis v lokalno podatkovno bazo. Pred vsakim klicem metode se preverijo nastavitve aplikacije, kjer mora biti omogočeno pošiljanje v oblak. V metodi podamo vrednosti vseh atributov razreda, ki predstavljajo podatke za pošiljanje. Ker želimo, da aplikacija nemoteno deluje naprej, ustvarimo novo nit, v kateri se izvede pošiljanje podatkov do storitve. V metodi `run` ustvarimo razred, ki predstavlja relacijo v oblaku, ter napolnimo podatke. Za pošiljanje tega razreda potrebujemo primernega upravljavca, ki poskrbi, da se vzpostavi povezava do prave relacije. Prek http sporočil se s pomočjo pretvornika vsi podatki pretvorijo v JSON obliko in tako pošljejo v storitev (koda 6).

```
final static void addCoordinate(final String id, final String longitude,
final  String latitude)
{
      //pridobivanje podatkov
      final CoordinateController uc = new CoordinateController();
      try {
        uc.create(c);
      } catch (Exception e) {
          return;
      }

      };
      checkUpdate.start();
}
```

Koda 6: Metoda za pošiljanje podatkov prek spletne storitve

Spletna storitev, ki črpa vire iz platforme v oblaku, ima tako dostop tudi do podatkovne baze, v katero zapiše podatke, ki jih prejme v zahtevku. V storitvi se podatki s pomočjo pretvornika pretvorijo v obliko za zapis. Ko se zapis izvede, storitev vrne odgovor, v katerem prejmemo podatke o uspešnosti vpisa. Slika spodaj (slika 7 prikazuje zapise v relaciji »coordinate«.

ID/Name	datum	idIntervencije	latitude	longitude
id=5644406560391168	21.08.2014 06:31:45	5	37.422005	121.084095
id=5654313976201216	21.08.2014 06:30:50	5	37.422005	-121.084095

Slika 7: Slika prikazuje zapise v podatkovni bazi GAE

5.4 Povezava s ponudnikom oblačnih storitev – Microsoft Azure

Poleg zgoraj predstavljenega ponudnika Google smo v našo aplikacijo integrirali tudi Microsoftovo ponudbo v oblaku. Kot ste lahko prebrali v poglavju, kjer je podrobna predstavitev ponudnikov, se njihova rešitev v oblaku imenuje Azure. V naslednjih podpoglavjih boste lahko prebrali, kako in kaj je potrebno storiti za integracijo Android mobilne aplikacije z rešitvijo Azure.

Kot smo predvidevali, je podjetje Microsoft tudi temu področju posvetilo veliko pozornosti in se zelo približalo razvijalcem. Glavno razvojno okolje za Microsoftove rešitve je program Visual Studio. Ko namestimo omenjeni program, je le-ta že pripravljen za povezavo z Azure storitvami. Tako lahko razvijemo spletno aplikacijo in jo naložimo neposredno v oblak. Polega tega niso zanemarili tudi drugih okolji, s katerimi lahko sobiva okolje Azure. Dostopni so literatura in razširitev za razvoj v okolju Eclipse ter operacijski sistem Android. To najdemo tudi za operacijske sisteme iOS, HTML/JavaScript itd.

5.4.1 Azure z uporabniškega vidika

Kot smo lahko prebrali, tudi Microsoft ponuja enomesečno brezplačno uporabo svoje platforme. Za možnost uporabe se moramo registrirati, poleg splošnih podatkov potrebujemo tudi kreditno kartico, katere podatke vnesemo ob prijavi. V času brezplačne ponudbe kartice ne bremenijo, vendar je s tem sistem že pripravljen za plačljivo licenco. Po preteku enega meseca moramo možnost zaračunavanja omogočiti. Če tega ne storimo do izteka, se aplikacije zamrznejo. Aplikacije niso dostopne, ohranjajo pa podatke, tako da jih lahko povrnemo v delovanje.

Tako kot Google tudi Azure z uporabnikom komunicira prek uporabniškega vmesnika. Z njegovo pomočjo lahko tako kot pri večini upravljamo s svojimi viri (računalniška moč,

podatkovna baza, storitve za neposredno sporočanje in komunikacijo z mobilnimi napravami). Prek njega tako ustvarjamo nove instance želenih storitev, ki jih uporabimo v aplikacijah. V našem primeru smo uporabili mobilno storitev, s katero omogočimo povezavo z mobilnimi napravami.

5.4.2 Strežniška storitev

Kot ste lahko prebrali v prejšnjem poglavju, smo morali pri GAE razviti storitev, ki smo jo naložili v oblak in je skrbela za delo z viri v oblaku. Pri Azure je v ta namen že razvita mobilna storitev, ki nadomešča naš lasten razvoj storitve za komunikacijo med mobilno aplikacijo in oblakom. Pri storitvi se prav tako kot pri GAE uporablja RESTful knjižnica.

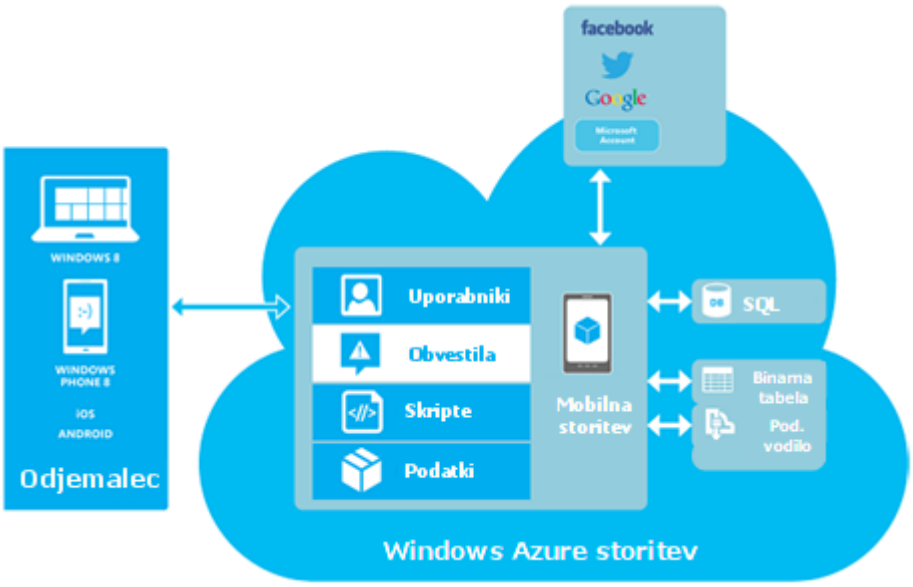
Storitev je aktivna, ko ustvarimo novo instanco. Storitev je na podlagi imena dosegljiva vsem odjemalcem in tako povezljiva z več različnimi odjemalci hkrati. Omenjali smo že, da ima storitev v ozadju tabelo, v katero lahko shranjuje različne podatke. Tabele so lahko poljubno oblikovane, pri čemer ni nujno, da so deklarirane v samem spletnem delu. Odjemalec lahko ob pošiljanju podatkov posreduje želene stolpce tabele.

S to rešitvijo je Microsoft razvijalcem precej olajšal delo, saj je zmanjšal težave pri razvoju storitve za delo s podatkovno bazo. Dobra stran je tudi ta, da je poleg pripravljenega paketa za Android podana tudi obsežna in nazorna dokumentacija za implementacijo in uporabo le-te.

Spodnja slika (slika 8) nam zelo nazorno prikazuje, kaj vse nam omogoča mobilna storitev. Naša aplikacija nastopa kot Android aplikacija, ki podatke prek mobilne storitve pošilja v tabele. Iz slike je razvidno tudi, da se prek omenjene storitve lahko povezujemo do preostalih zmožnosti Azure oblaka.

5.4.3 Integracija storitve v aplikacijo - odjemalec

Ko smo opisovali integracijo z Googlovim oblakom, smo omenili, da ga bomo uporabili kot arhiv za podatke iz naše aplikacije. Za uspešno delovanje moramo najprej vzpostaviti povezavo med odjemalcem in storitvijo, kasneje pa je potrebno določiti še, katere podatke bomo posredovali v zahtevkih. To smo določili z novim razredom, ki predpisuje tudi obliko tabele v oblaku.



Slika 8: Slika prikazuje zmožnosti mobilne storitve v oblaku Azure. Na podlagi: https://msdn.microsoft.com/en-us/library/azure/jj554228.aspx

Kot že omenjeno, je Microsoft pripravil pakete, ki jih enostavno vključimo v našo aplikacijo in tako že lahko uporabljamo storitev v oblaku. Za Android obstaja nabor arhivskih Java datotek (jar datoteke). Jar datoteka je skupek Javanskih razredov z dodanimi meta podatki in ostalimi datotekami, ki jih potrebujemo, da celoten paket deluje tako, kot je predvideno (slike, opisi itd). S takim pristopom je omogočena lažja distribucija razredov. [14] Ko takšen paket vključimo v svoj Android projekt/aplikacijo, lahko te razrede uporabimo pri razvoju.

Arhivske Java datoteke smo vključili v aplikacijo, kar nam je omogočilo, da smo dodali razred, ki omogoča delo z mobilno spletno storitvijo. Najprej moramo deklarirati spremenljivko tipa *MobileServiceClient,* s katero omogočimo dostop do spletne storitve. Povezavo vzpostavimo z uporabo enoličnega ključa, ki nam ga dodeli storitev. Spodnja koda prikazuje, kako se naša aplikacija povezuje s storitvijo (koda 7). Ko je povezava vzpostavljena, sta mogoča pošiljanje podatkov in komuniciranje z oblakom.

5.4.4 Uporaba storitve

Za uporabo storitve so morali biti izpolnjeni pogoji, ki so opisani v zgornjih podpoglavjih. V kodi smo pripravili vse potrebno, da se lahko podatki prek storitve pošiljajo v tabele, ki jih definiramo programsko. Mobilna storitev predvideva, da z atributi razreda določimo stolpce tabele. Razredi, ki opredelijo stolpce tabele, so enaki kot pri GAE (koda 7).

```
private static MobileServiceClient mClient;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		//ostali del kode

		try {
			mClient = new MobileServiceClient("https://fireforalert.azure-
mobile.net/","WKIkpaimzwijeZZETHjfQtyUIlagga50",this);
		} catch (MalformedURLException e) {
			Log.e("mClient", e.toString());
		}
	}
```

Koda 7: Prikaz povezave z Azure mobilno storitvijo.

Ko se ustvari nov dogodek, se po zapisu v lokalno podatkovno bazo podatki shranijo tudi v oblak. Na podlagi izbranega ponudnika v nastavitvah aplikacije se izvede zapis v podatkovno bazo. Če je izbran Azure, se izvede spodnja koda (koda 8), iz katere je razvidno, da je vstavljanje preprost klic metode `getTable`, ki vrne iskano tabelo. Za tem lahko že kličemo `insert` oz. `update` v tabelo. V to metodo podamo prej omenjene razrede, kjer so s podatki napolnjeni atributi razreda.

S spodnjim klicem se v ozadju prek http zahteve izvede pošiljanje podatkov do mobilne storitve. Storitev poskrbi, da pošlje podatke do storitve, ki upravlja s podatkovno bazo in jih zapiše v tabelo. Tako kot pri vseh ponudnikih imamo tudi tu možnost vpogleda v tabele, kamor se poslani podatki shranjujejo.

```
final static void addCoordinateAzure(final String id, final String longitude,
final String latitude)
    {
      Coordinate c = new Coordinate();
      c.idIntervencije = id;
      c.latitude = latitude;
      c.longitude = longitude;
      c.datum = dateFormat.format(date);

      mClient.getTable(Coordinate.class).insert(c, new
TableOperationCallback<Coordinate>() {
        public void onCompleted(Coordinate entity, Exception exception,
ServiceFilterResponse response) {
            if (exception == null) {
                Log.e("mClient", "uspešno dodano");
            } else {
                Log.e("mClient", exception.toString());
```

Koda 8: Prikaz klica v stavljanje v tabelo Azure s pomočjo mobilne storitve

Poglavje 6 Sklepne ugotovitve

V diplomski nalogi smo preučili večje število ponudnikov, ki ponujajo raznovrstne rešitve v oblaku. Opazimo lahko, da je ponudba na tem področju zelo raznovrstna, tako da med njimi lahko najdemo takšno, ki nam najbolj ustreza. Odločimo se lahko za takšno, kjer lahko izberemo vse po svojih potrebah. Tak primer je na primer ponudnik Heroku, vse ostale pa temeljijo na tako imenovani paketni ponudbi. V tem primeru zakupimo neko količino, ki je morda pri svojem končnem izdelku ne bomo mogli izkoristiti. Tu lahko opazimo težavo, saj tako plačujemo za nekaj, česar ni moč izkoristiti. Zanimivo je tudi to, da je Google šele v mesecu avgustu omogočil uporabo SQL podatkovne baze, ostali ponudniki pa so imeli to vključeno že na začetku.

Aplikacija, ki je razvita za operacijski sistem Android, je razvita do načrtovanih ciljev. Poleg sprejemanja in preverjanja prispelih sporočil beleži tudi same geografske pozicije telefona med potekom dogodka. Z vsemi zabeleženimi podatki lahko pripravimo podrobnejšo analizo in potek dogodka, kar je za gasilske intervencije pomembno. Nenazadnje je aplikacija povezana tudi s storitvijo v oblaku, kjer so vsi ti podatki arhivirani.

Primerjali smo Googlovo in Microsoftovo platformo, med katerima lahko v naši aplikaciji izbiramo. Za arhiviranje podatkov v Google app engine podatkovno strukturo je bilo potrebno najprej razviti spletno storitev. Tu smo naleteli na novost in pa tudi na težavo pri razvoju, saj smo storitev razvijali v REST arhitekturi. Posledica tega je tudi ta, da je bilo potrebno v aplikacijo vključiti več razredov, ki so opredelili komunikacijo in obliko zahtevkov do storitve. Pri Azure pa takšnih težav nismo imeli, saj obstaja storitev, ki je že pripravljena za uporabo. V razvojni program smo najprej vključili knjižnico za delo s storitvijo. Potem pa smo uporabili metode, ki opredeljevale vse, kar smo pri Google app engine morali sami. Microsoftova platforma se je izkazala za boljšo in hitreje povezljivo z Android aplikacijo.

Aplikacijo bi bilo mogoče nadalje razviti in izboljšati v več smereh. Tako bi lahko omogočili sledenje in sprejemanje več dogodkov. Na mestu bi bila tudi dodelava s predstavitvijo koordinat, ki jih pridobivamo med potekom dogodka. Predstavili bi jih lahko na zemljevidu, kjer bi nam bilo na primer omogočeno tudi prikazovanje poti, ki smo jo pri dogodku opravili.

V spletnem delu bi lahko razvili portal, ki bi obdeloval in prikazoval podatke, uporabili pa bi tudi zemljevid, prikazovanje poti, koliko časa je dogodek trajal itd.

V splošnem smo ponovili znanje s področja programiranja in razvijanja aplikacij v operacijskem sistemu Android. Na novo smo se spoznali s povezovanjem v oblak in kako se v ta namen razvija spletne storitve, ki so danes pogosto uporabljane tudi v praksi. Z diplomskim delom smo tako pridobili nova znanja, ki jih bomo lahko uporabili v vsakdanjem življenju in nadaljnji računalniški poti.

Literatura

[1] Kris Jamasa, "Cloud computing", *SaaS, PaaS, IaaS, Virtualization, Business Models, Mobile, Security and more,* 2013.

[2] Wei-Meng Lee, "Begining Android Application Development", 2011.

[3] Jerome Louvel, Thierry Templier, Thierry Boileau, "Restlet in action", *Developing RESTful web APIs in Java,* 2013

[4] (2014, maj) Uradna spletna stran programa Eclipse. Dostopno na: https://www.eclipse.org/org/

[5] (2014, maj) Spletna stran o Android OS. Dostopno na: http://sl.wikipedia.org/wiki/Android_(operacijski_sistem)

[6] (2014, maj) Spletna stran, ki opisuje Android SDK. Dostopno na: http://en.wikipedia.org/wiki/Android_software_development

[7] (2014, maj) Spletna stran z razlago Android emulatorja. Dostopno na: http://developer.android.com/tools/help/emulator.html

[8] (2014, maj) Članek organizacije NIST z naslovom *The NIST Definition of Cloud Computing,* objavljen leta 2011. Najdeno na: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[9] (2014, julij) Vsebina o računalništvu v oblaku. Dostopno na: http://en.wikipedia.org/wiki/Cloud_computing

[10] (2014, julij) Uradna spletna stran z obrazložitvijo gradnikov za uporabniški vmesnik. Dostopno na: http://developer.android.com/guide/topics/ui/index.html

[11] (2014, julij) Spletna stran z opisom načina shranjevanja podatkov. Dostopno na: http://developer.android.com/training/basics/data-storage/index.html

[12] (2014, julij) Spletna stran, ki opisuje razred broadcast receivers in njegovo uporabo. Dostopno na: http://www.tutorialspoint.com/android/android_broadcast_receivers.htm

[13] (2014, november) Spletna stran za uporabo knjižnice Objectify. Dostopno na: https://code.google.com/p/objectify-appengine/

[14] (2014, november) Spletna stran, ki opisuje, kaj so arhivske Java datoteke oziroma kaj pomeni kratica jar. Dostopna na: http://en.wikipedia.org/wiki/JAR_(file_format)